CS-2303 System Programming Concepts WPI, A-term 2017  
Professor Mike Ciaraldi Quiz #3 (20 points)  
Quiz date: Thursday, September 14, 2017 ANSWER KEY

|  |  |  |
| --- | --- | --- |
| Question | Possible | Points |
| 1 | 5 |  |
| 2 | 5 |  |
| 3 | 5 |  |
| 4 | 5 |  |
| Maximum | 20 |  |

NAME:

WPI E-mail ID:

READ THESE INSTRUCTIONS BEFORE STARTING THE QUIZ.

This is an open-book, open-notes quiz. You can use a computer or e-reader to access your notes and/or book stored on that device. You are not allowed to access the network during the quiz.

This quiz is worth 20 points. Answer questions in the spaces provided on the quiz itself. Take the number of points assigned to each question and the amount of space provided for your answer as a measure of the length and difficulty of the expected solution.

Be sure to answer the question which is actually being asked, in a way which demonstrates that you understand the meaning of the question and the answer. For example, if the question asks you to say what happens and why, you have to tell both to get full credit. If the question asks you to explain something in general and to give an example, you have to do both to get full credit. *Remember, the graders know the answers to these questions. What you write has to show that you understand the question and its answer.*

A *program fragment* is a section of code which is part of a complete program. You can make any reasonable assumptions about the rest of the program.

All questions apply to C unless otherwise indicated.

This quiz will end at 8:20 pm. *It is* STRONGLY *suggested that you read the entire quiz before attempting to answer any questions. Also, re-read all the parts of any one question before you start answering it.*

1. **Memory allocation.** [5 points]
   1. List two differences between the functions calloc() and malloc(). [2 points]

*Answer: You call malloc() with one argument, the number of bytes you want; you call calloc() with two, the number of elements you want and the size of each element. calloc() fills the allocated memory with zeroes; malloc() does not initialize it at all.*

* 1. Since malloc() and calloc() return void\*, you generally have to cast this result. You have to do this if you are going to use the pointer immediately. Or if you are going to save it for later, you have to do this so it matches the type of variable into which it is being stored. Give two reasons why the compiler needs to know the type of a pointer. [2 points]

*Some possible answers: So it knows how many bytes of memory to access. So it knows how to handle the particular type of data (or whatever) stored there; for example, the instructions for handling integers are different than the ones for floats. So it knows how to do pointer arithmetic; for example, what number to add to the address when you use the ++ operator. To be able to find fields inside a struct or object (object are only in C++, not C).*

* 1. Why is it that all pointers on a given machine are usually the same size? [1 point]

*Answer: Any pointer has to be able to hold any valid address. In other words, since the compiler does not know ahead of time where in memory things will be stored, it has to make every pointer big enough to hold any address.*

1. **Run-Time Errors.** [5 points]
   1. In general, what does it mean when your program produces a segmentation fault (segfault)? [2 points]

*Answer: The program tried to access an address in memory which it was not allowed to access.*

* 1. List three possible specific reasons why your program might signal a segfault. In other words, do not merely repeat your answer to part a; give examples of what could cause that to occur. [3 points]

*Some possible answers: Moved off the end of an array. Null pointer. Uninitialized pointer. Note: trying to use a pointer to a block of memory which has been freed is invalid, but will probably not by itself cause a segfault, because the pointer points to a section of memory which the program is allowed to use. Note:   
Saying that the pointer is not valid is not specific enough.*

1. **Array Access.** [5 points]
   1. Suppose you have an array declared **int a[10]**, and your program needs to access element **a[i]**. What sequence of operations does the program perform to determine at run-time where to find that piece of data? [3 points].

*Answer. It takes the value of the index (i), multiplies it by the length of one element of the array (in this example, 4), and adds that to the address of the base of the array (i.e., the address of a[0]). Note: Despite what some students think, to find the address of element a[i] the computer does NOT have to start with the address of element a[0] every time, and then keep incrementing the pointer until it reaches a[i].*

* 1. If your program is going to move through the elements of an array in order, it can be more efficient to initialize a pointer into the array and then keep incrementing the pointer using the ++ operator, as compared to using an index into the array (such as **a[i]**). Why is that? [2 points]

*Answer: Using an index requires the processor to perform the sequence of operations listed in the previous answer. Incrementing just requires adding one constant, which is a faster operation than multiplying and adding.*

1. **Make and Makefiles.** [5 points]

Suppose you have the following makefile. Note: The line numbers do not appear in the file; they are just there to make it easier to write your answers. You can assume that the file is properly formatted; for example, the indented lines are indented with a TAB character.

*1.* **all: foo**

*2.*

*3.* **foo: foo.o util.o**

*4.* **gcc –g –Wall foo.o util.o –o foo**

*5.*

*6.* **foo.o: foo.c util.h**

*7.* **gcc –g –Wall foo.c**

*8.*

*9.* **util.o: util.c util.h**

*10.* **gcc –g –Wall util.c**

*11.*

*12.* **clean:**

*13.* **rm –f \*.o foo**

1. Suppose you type the command **make clean** . Which line or lines will be executed, in what order, and why? [2 points]

*Answer: Line 13, because that is the command associated with the rule “clean:”*

1. Suppose after that, you modify the file foo.c, and type the command **make** . Which line or lines will executed, in what order, and why? [3 points]

*Answer: Here is a very detailed answer. Any answer which contains enough information to show that you understand the question was considered correct. Note: Remember, you just did a “make clean” so the executable file foo and all the .o files have been deleted.*

*First, because you only typed “make” and did not specify a rule, the make utility looks at the first rule, which is “all: foo”. So it knows it needs to create an up-to-date foo.*

*So then it looks at line 3, and sees that it needs foo.o and util.o, which do not exist. So it recursively looks for rules for creating those two files.*

*It sees line 6, which says that foo.o depends on foo.c and util.h . The file foo.o does not exist, but the other two do, so it executes the command on line 7. Executing the gcc command on file foo.c, with option –c, causes the source file to be compiled, producing foo.o, so the rule has been satisfied.*

*For the same reasons, the make utility sees line 9 and executes the command on line 10, to produce the object file foo.o . Note: The utility is free to execute line 7 before line 10, as long as both are complete before it tries to finish satisfying the rule on line 3.*

*Then the make utility sees that the dependencies of the rule on line 3 now exist, but the file to the left of the colon does not. So it executes the command on line 4. That command runs the linker program, which takes the .o files, links them with functions from the libraries, and produces the executable file foo .*

*Note: If you had run the make command, then modified the file foo.c, and then ran make again, what would happen? On the second make, the make utility would have first executed line 7, because foo.o depends on foo.c, and foo.c is newer. It would not have to execute line 10, because util.o would already be newer than util.c (because of the previous make). Then it executes line 4, because foo depends on foo.o, which has now become newer.*